Ontology verbalisation hands-on exercise @ JOWO 2022

Zola Mahlaza

The main objective of this exercise is to introduce participants to the task of generating natural language text from an ontology. Participants will be tasked with designing templates for generating English, German, and isiZulu[1] text. The three languages have been chosen chosen in order to expose participants to challenges, or lack thereof, that may be due to the complexity of each output language's grammar.

To complete this exercise, participants must form groups of 2-4 people, if possible.

1 Exercise outcomes

Participants are expected to possess the ability to execute the following tasks after completing the hands-on exercise:

- Design an English template to verbalise an OWL axiom
- Aggregate multiple English sentences to improve paragraph readability
- Ideate on how to obtain lexical resources for classes in an ontology
- Design a German template to verbalise an OWL axiom
- Design an isiZulu template to verbalise an OWL axiom

2 Optional software dependencies

There are optional programming tasks that require a number of dependencies. Should each team wish to also attempt the programming tasks then there must be at least one team member who has an IDE installed and who has some knowledge of the following:

- Java 11 (or later)
- OWLAPI (`https://github.com/owlcs/owlapi`)

Knowledge of the OWLAPI is not strictly necessary as most of the OWLAPI-related functionality has been implemented. However, such knowledge may assist when debugging the code. The only dependency that is absolutely necessary in order to be able to do the optional programming tasks is Java.

[1]The most prevalent South African language by L1 speakers. `https://en.wikipedia.org/wiki/Zulu_language`

3 Generating English text

The first batch of tasks focuses on designing templates for verbalising an ontology in English. Specifically, we will focus on a subset of the OntoVerbal verbaliser [3, 4, 2].

The OntoVerbal(-M) verbaliser was created by Liang et al. [3, 4, 2] for generating English and Mandarin. It was intended to be used as an assistive tool by individuals who want to audit the contents of ontologies but have no expertise in either ontologies or ontology languages.

Consider the following axioms, given in functional syntax, pertaining to the Valve class taken from Liang et al. [4]:

- `DisjointClasses(AnatomicalCavity Valve)`
- `SubClassOf(TricuspidValve Valve)`
- `SubClassOf(PartialValve Valve)`
- `SubClassOf(Valve AnatomicalConcept)`
- `SubClassOf(SemiLunarValve Valve)`
- `SubClassOf(VestigialCardiacValve Valve)`
- `SubClassOf(MitralValve Valve)`
- `EquivalentTo(AtrioVentricularValve ObjectUnionOf(Valve ObjectSomeValues-From( hasValveInput AtriumCavity) ObjectSomeValuesFrom(hasValveOutput Ven-tricularCavity)))`

The verbaliser has the ability to take these axioms and transform them to produce the following text, as shown by Liang et al. [4]:

```
A valve is a kind of anatomical concept. More specialised
kinds of valve are mitral valve, partial valve, semi lunar valve,
tricuspid valve and vestigial cardiac valve. Also, a valve is
different from an anatomical cavity. Another relevant aspect
of a valve is that an atrio ventricular valve is defined as a valve
that has valve input an atrium cavity and has valve output a
ventricular cavity.
```

While the original OntoVerbal has the capacity to verbalise a variety of axioms, we limit ourselves to the verbalisation of "simple axioms" [3, pg340] for this tutorial. We say that an axiom is simple in the context of OntoVerbal if it is one of the following $A \sqsubseteq B$, $A \equiv B$, $A \sqsubseteq \neg B$, and $a \in A$ where A and B are atomic classes.

To demonstrate the capability, and limitations, of the exercise version of the OntoVerbal verbaliser, let us look at the output it produces when it is given the axioms pertaining to the `VegetarianTopping` class from the pizza tutorial ontology[2]:

[2] `https://protege.stanford.edu/ontologies/pizza/pizza.owl`

```
A(n) VegetarianTopping is a kind of DomainConcept, Thing, Food, and PizzaTopping. More specialised kinds of VegetarianTopping are CaperTopping, VegetableTopping, CheeseyVegetableTopping, TomatoTopping, SlicedTomatoTopping, ParmesanTopping, RocketTopping, RosemaryTopping, ArtichokeTopping, GorgonzolaTopping, TobascoPepperSauce, LeekTopping, FruitTopping, CajunSpiceTopping, NutTopping, CheeseTopping, MushroomTopping, PeperonataTopping, MozzarellaTopping, SweetPepperTopping, RedOnionTopping, GarlicTopping, SultanaTopping, OliveTopping, IceCream, OnionTopping, AsparagusTopping, PetitPoisTopping, SauceTopping, HerbSpiceTopping, PepperTopping, JalapenoPepperTopping, SundriedTomatoTopping, GoatsCheeseTopping, SpinachTopping, FourCheesesTopping, GreenPepperTopping, HotGreenPepperTopping, and PineKernels. A(n) VegetarianTopping is defined as VegetarianTopping. In addition, a(n) VegetarianTopping is different from a(n) CheeseyVegetableTopping, a(n) Spiciness, a(n) PizzaBase, a(n) ParmaHamTopping, a(n) Rosa, a(n) QuattroFormaggi, a(n) PrawnsTopping, a(n) PrinceCarlo, a(n) ThinAndCrispyPizza, a(n) MeatTopping, a(n) AnchoviesTopping, a(n) MixedSeafoodTopping, a(n) Napoletana, a(n) HotSpicedBeefTopping, a(n) Mushroom, a(n) SloppyGiuseppe, a(n) AmericanHot, a(n) MeatyPizza, a(n) HamTopping, a(n) Fiorentina, a(n) Caprina, a(n) Pizza, a(n) DeepPanBase, a(n) NonVegetarianPizza, a(n) InterestingPizza, a(n) UnclosedPizza, a(n) SpicyPizza, a(n) ThinAndCrispyBase, a(n) Mild, a(n) Parmense, a(n) FishTopping, a(n) Capricciosa, a(n) Cajun, a(n) PeperoniSausageTopping, a(n) Margherita, a(n) Giardiniera, a(n) FruttiDiMare, a(n) LaReine, a(n) Soho, a(n) IceCream, a(n) CheeseyPizza, a(n) Hot, a(n) ChickenTopping, a(n) NamedPizza, a(n) SpicyPizzaEquivalent, a(n) FourSeasons, a(n) VegetarianPizza, a(n) Medium, a(n) PolloAdAstra, a(n) American, a(n) VegetarianPizzaEquivalent2, a(n) VegetarianPizzaEquivalent1, a(n) Siciliana, a(n) Veneziana, and RealItalianPizza.
```

Additional output examples can be found at https://people.cs.uct.ac.za/~zmahlaza/jowo2022/en/ontoverbaloutput.html. You are free to download the jar file with the exercise version of the verbaliser at https://people.cs.uct.ac.za/~zmahlaza/jowo2022/en/ontoverbalforrunning.zip and run it with you preferred ontology or the ones listed at http://www.meteck.org/MoReNL/JOWO22tutorialOntologies/.

3.1 Tasks

For purposes of improving the verbaliser, you will design your own templates and implement them in Java. The implementation of the templates in Java is optional and those interested in completing the Java verbaliser should download the skeleton source code from https://people.cs.uct.ac.za/~zmahlaza/jowo2022/en/verbalisercode.zip.

1. For each of the following axiom type $A \sqsubseteq B$, $A \equiv B$, $A \sqsubseteq \neg B$, and $a \in A$ where A and B are atomic classes, execute the following tasks:

   (a) Design a template for presenting the axiom in English. Use a piece of paper to explore the various possibilities and write down example sentences they can produce.

   (b) Create a declarative form of your template in a text file. For instance, if you use Mustache syntax[3] then you can represent slots as {{`slotName`}} while all other text is fixed. The example template `Hello` {{`participantName`}} that uses Mustache syntax can be used to generate texts of the form `Hello Aubrey`, `Hello Adam`, etc. where `Aubrey` and `Adam` are possible participant names. Alternatively, try to use the proposed syntax for the template language for Abstract Wikipedia at `https://meta.wikimedia.org/wiki/Abstract_Wikipedia/Template_Language_for_Wikifunctions`, or make up your own one.

   (c) Complete the `verbaliseSimpleAxiom` method found in the downloaded skeleton code (in `OntoVerbal.java`) to use the newly created declarative template to generate a sentence for each axiom type. [**Optional task**]

2. All axioms of a certain type can be aggregated into a single set based on the role played by a certain class. Using the axioms from the `Valve` example that has already been presented, it is possible to form a set of all subsumption axioms that include the `Valve` as the superclass and obtain the set $\{TricuspidValve \sqsubseteq Valve, PartialValve \sqsubseteq Valve\}$. We now need to design templates that can present such sets using English. Execute the following tasks:

   (a) Given classes A and $B_1, ..., B_n$, design a template for verbalising the set of all subsumption axioms $B_i \sqsubseteq A$ where A is fixed for all Bs $(1 \leq i \leq n)$.

   (b) Given classes A and $B_1, ..., B_n$, design a template for verbalising the set of all subsumption axioms $A \sqsubseteq B_i$ where A is fixed for all Bs $(1 \leq i \leq n)$.

   (c) Given classes A and $B_1, ..., B_n$, design a template for verbalising the set of all equivalence axioms $A \equiv B_i$ where A is fixed for all Bs $(1 \leq i \leq n)$.

   (d) Given classes A and $B_1, ..., B_n$, design a template for verbalising the set of disjoint axioms $A \sqsubseteq \neg B_i$ where A is fixed for all Bs $(1 \leq i \leq n)$.

   (e) Given the class A, design a template for verbalising the set of all individuals of A.

3. For each of the templates designed in Task 2, create a declarative form of your template in a text file. [**Optional task**]

4. For each of the declarative templates created in Task 3, use them to complete the `verbaliseSimpleAxioms` method found in `OntoVerbal.java` code. [**Optional task**]

5. Generating a paragraph from the previous sentences:

   (a) Design a template to generate a paragraph that aggregates the sentences produced by the templates you designed in Task 2.

   (b) Create a declarative form of the aggregate template in a text file. [**Optional task**]

[3]`https://mustache.github.io/`

(c) Complete the `generateAggregatedSentences` method found in `OntoVerbal.java` code to use your declarative template and generate the final paragraph. [**Optional task**]

The example text listed above that was generated by the exercise version of the OntoVerbal verbaliser is readable, however, it does not look completely natural. Let us now consider how to improve the quality of the verbaliser's output text.

In the verbalised text, we rely on the name of the class found in the URI. Specify one or two alternate resources/locations you can use to retrieve the name of each class.

4 Generating German text

One is unlikely to encounter a lot of grammatical complexity when designing templates for English. However, the same cannot be said when designing templates for languages such as German. To the best of knowledge, the most recent and published verbaliser[4] that supports German relies on Grammatical Framework [6] – a language and environment for parsing and generating natural language text.

For this part of the tutorial, you will explore some of the linguistic challenges that arise for German for even basic axiom types. You will be tasked to design a template for a single axiom. (Unlike the previous section, we will not explore the creation of code to generate the text. Participants who are interested in the use of Grammatical Framework for capturing their templates can do so in their own time.) If none of you group members speak German, you may try Spanish (which will illustrate certain challenges sufficiently as well, although not as much as with German).

4.1 Tasks

1. Consider axioms of the kind $A \sqsubseteq \exists R.B$ and design a template for presenting such axioms in German. Use a piece of paper to explore the various possibilities and decide on your best template.

2. Write down the output that will be produced by your chosen template for each of the following axioms:

   (a) $Citizen \sqsubseteq \exists loves.BlackDog$

   (b) $RockMusician \sqsubseteq \exists loves.BlackGoose$

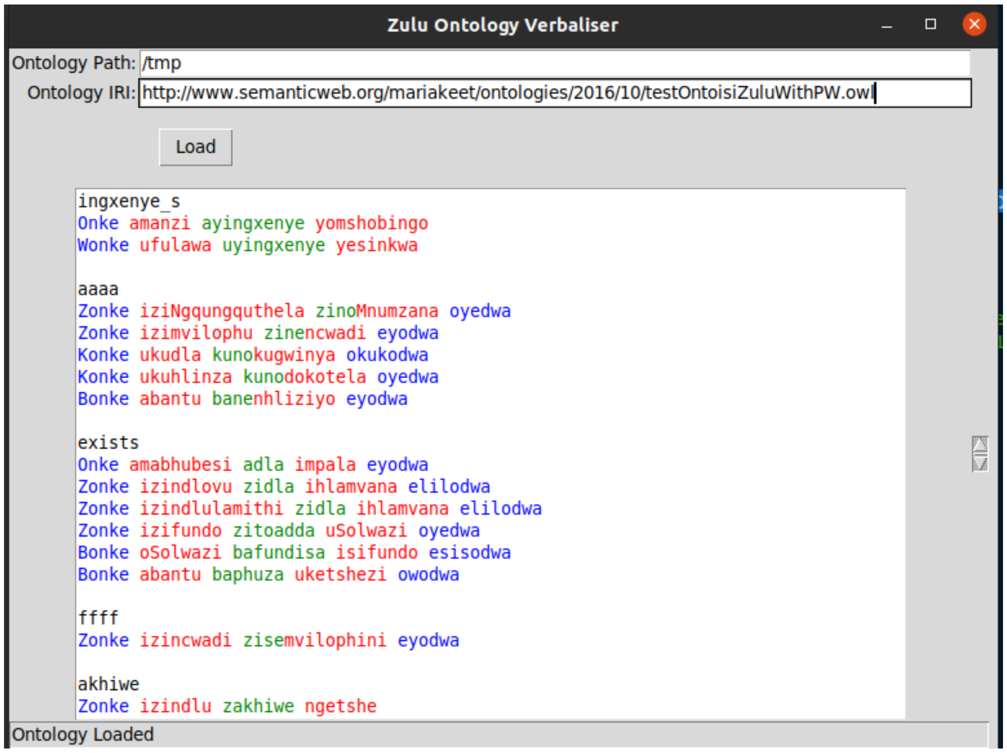
3. Each group must share their output for Task 2 with at least one other group and decide on which output is best, if any.

[4]`https://github.com/Attempto/ACE-in-GF`

5 Generating isiZulu text

Some of the challenges that arise when trying to design templates for German also exist for other languages and they may compounded for other languages. You will now explore this by examining the design of templates for isiZulu.

The isiZulu verbaliser [1] was designed to be the first natural language generator for ontologies that supports an African language. IsiZulu is member of the Nguni language group and it is primarily spoken in Southern Africa. Like other Niger-Congo B languages, it has complex morphology and its agreement system makes verbalisation challenging. The isiZulu verbaliser is multi-platform as its written in Python and has the ability to take supported axioms and transform them to produce text as shown in Figure 1.



Ontology Path: /tmp
Ontology IRI: http://www.semanticweb.org/mariakeet/ontologies/2016/10/testOntoisiZuluWithPW.owl

```
ingxenye_s
Onke amanzi ayingxenye yomshobingo
Wonke ufulawa uyingxenye yesinkwa

aaaa
Zonke iziNgqungquthela zinoMnumzana oyedwa
Zonke izimvilophu zinencwadi eyodwa
Konke ukudla kunokugwinya okukodwa
Konke ukuhlinza kunodokotela oyedwa
Bonke abantu banenhliziyo eyodwa

exists
Onke amabhubesi adla impala eyodwa
Zonke izindlovu zidla ihlamvana elilodwa
Zonke izindlulamithi zidla ihlamvana elilodwa
Zonke izifundo zitoadda uSolwazi oyedwa
Bonke oSolwazi bafundisa isifundo esisodwa
Bonke abantu baphuza uketshezi owodwa

ffff
Zonke izincwadi zisemvilophini eyodwa

akhiwe
Zonke izindlu zakhiwe ngetshe
```

Ontology Loaded

Figure 1: Sample of texts generated by the isiZulu verbaliser

We demonstrate the challenge of verbalising axioms in the language by using output that is produced by the OWLSIZ verbaliser [5]. When the verbaliser is fed the axiom subClassOf(herb plant), it generates the question Ingabe **lo**nke <u>ihebhu</u> **ling**<u>umuthi</u>? "Is every herb a plant?" where underlined sections are the isiZulu class names. The sections that are highlighted in bold are automatically generated and may change based on the value of the classes. Technically, the values change as a result of the noun class of the OWL class' lexical item (see [7], for instance, for additional details on the isiZulu noun class system).

5.1 Tasks

Since participants may not be familiar with isiZulu, let us start by reverse engineering a possible English template to an isiZulu one.

1. Suppose we verbalise $A \sqsubseteq B$ using the template `Every [A] is a [B]` in English. Use Google Translate[5] to devise an isiZulu template.

2. Use the template designed in Task 1 to generate what you believe would be the output for the following axioms:

   (a) $Dog \sqsubseteq Animal$ (with isiZulu labels: $Inja \sqsubseteq Isilwane$)

   (b) $Student \sqsubseteq Person$ (with isiZulu labels: $Umfundi \sqsubseteq Umuntu$)

   (Hint: You can consult the following page `https://en.wiktionary.org/wiki/Appendix:Zulu_concords`)

References

[1] C. Maria Keet, Musa Xakaza, and Langa Khumalo. Verbalising OWL ontologies in isizulu with python. In Eva Blomqvist, Katja Hose, Heiko Paulheim, Agnieszka Lawrynowicz, Fabio Ciravegna, and Olaf Hartig, editors, *The Semantic Web: ESWC 2017 Satellite Events - ESWC 2017 Satellite Events, Portorož, Slovenia, May 28 - June 1, 2017, Revised Selected Papers*, volume 10577 of *Lecture Notes in Computer Science*, pages 59–64. Springer, 2017.

[2] Shao Fen Liang, Donia Scott, Robert Stevens, and Alan L. Rector. Ontoverbal: a generic tool and practical application to SNOMED CT. *CoRR*, abs/1312.2798, 2013.

[3] Shao Fen Liang, Robert Stevens, Donia Scott, and Alan L. Rector. Automatic verbalisation of SNOMED classes using ontoverbal. In Mor Peleg, Nada Lavrac, and Carlo Combi, editors, *Artificial Intelligence in Medicine - 13th Conference on Artificial Intelligence in Medicine, AIME 2011, Bled, Slovenia, July 2-6, 2011. Proceedings*, volume 6747 of *Lecture Notes in Computer Science*, pages 338–342. Springer, 2011.

[4] Shao Fen Liang, Robert Stevens, Donia Scott, and Alan L. Rector. Ontoverbal: a protégé plugin for verbalising ontology classes. In Ronald Cornet and Robert Stevens, editors, *Proceedings of the 3rd International Conference on Biomedical Ontology (ICBO 2012), KR-MED Series, Graz, Austria, July 21-25, 2012*, volume 897 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[5] Zola Mahlaza and C. Maria Keet. OWLSIZ: An isiZulu CNL for structured knowledge validation. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 15–25, Dublin, Ireland (Virtual), 12 2020. Association for Computational Linguistics.

[5] `https://translate.google.com`

[6] Aarne Ranta. *Grammatical framework: Programming with multilingual grammars*. CSLI Publications, Center for the Study of Language and Information Stanford, 2011.

[7] Edith Khanyisile Twala. The noun class system of IsiZulu. Master's thesis, University of Johannesburg, 1992.